IEEE standards for FLOATING POINT NUMBERS

Excerpted from a document by Robert P. Webber, Longwood University

There are many slightly different floating point formats. In an effort to bring order from chaos, the Institute of Electrical and Electronics Engineers (IEEE) developed a standard form called **IEEE 754 single precision**. Many computers use this standard, and it is the one we will examine.

The IEEE standard uses 32 bits for each floating point number, divided into three fields.

- The left most bit is the sign bit.
- The next eight bits hold the exponent.
- The final 23 bits contain the fractional part.

Sign (1 bit)	Exponent (8 bits)	Fractional part (23 bits)

The sign bit is 0 for a positive number or zero, and 1 for a negative number.

The fractional part assumes the number is binary and in the form $1.xx\ldots x$, where $xx\ldots x$ denotes binary digits. This is called **normalized form**. Since the fractional part always begins with 1, there is no need to store that bit. Only the part after the binary point is stored. The assumed 1 is called a **hidden bit**, and it provides 24 bits of accuracy in only 23 bit spaces.

The exponent must allow for a sign, since exponents can be positive or negative. It must also allow for quick comparison to other exponents, because many comparisons must be done in floating point arithmetic. Two's complement notation would provide the sign, but not quick comparison. To allow that, the IEEE form uses **excess 127 notation**. In base 10,

Excess 127 exponent = signed decimal exponent + 127 .

For example, convert the decimal number 50.5 to IEEE format.

Step 1: Convert 50.5 to base 2.

$50.5 = 32 + 16 + 2 + ½ = 2^5 + 2^4 + 2^1 + 2^{-1} = 110010.1_2$.

Step 2: Write the number in normalized form. We must move the binary point five places to the left, so the exponent is 5.

$110010.1 = 1.100101 * 2^5$.

The fractional part is 100101. We drop the left-most 1, because it is the hidden bit. Notice there is no need to write trailing zeros when we write the number by hand. When we write it as the computer will store it, however, we will need to add 17 trailing zeros to make 23 bits for the fractional part in all. The complete fractional part is

10010100000000000000000 .

Step 3: Find the excess 127 form of the exponent.

$127 + 5 = 132 = 128 + 4 = 2^7 + 2^2 = 10000100_2$

The sign is 0, since the number is positive; the excess 127 exponent is 10000100, and the fractional part is 100101 followed by 17 zeros. The IEEE form is

01000010010010100000000000000000 ,

and this is how the number would be stored in the computer. We could write this as 424A0000 in hexadecimal form for better readability.

The form may be clearer if we break it into fields.

0	10000100	10010100000000000000000
sign	exponent	fractional part

Here's another example. Write -121.75_{10} in IEEE floating point format.

Step 1: Convert 121.75 to binary.

$121.75 = 64 + 32 + 16 + 8 + 1 + ½ + ¼$

$= 2^6 + 2^5 + 2^4 + 2^3 + 2^0 + 2^{-1} + 2^{-2} = 1111001.11_2$

Step 2: Write the binary number in normalized form.

$1111001.11 = 1.11100111 * 2^6$

Step 3: Find the excess 127 exponent.

$127 + 6 = 133 = 128 + 4 + 1 = 2^7 + 2^2 + 2^0 = 10000101_2$

The sign is 1, the fractional part is 11100111 followed by 15 zeros (for a total of 23 bits), and the exponent is 10000101. The IEEE form is

11000010111100111000000000000000

or C2F38000 in hexadecimal. Broken up into fields, it is

1	10000101	11100111000000000000000

sign exponent fractional part

Finally, here's an example with a negative exponent. Write 0.625_{10} in IEEE floating point format.

First, convert the number to binary and put it in normalized form.

$$0.625 = 5/8 = 1/2 + 1/8 = 2^{-1} + 2^{-3} = 0.101_2 = 1.01 * 2^{-1}$$

Next, write the exponent in excess 127 form.

$$-1+127 = 126 = 01111110_2$$

The number itself is positive, so the sign bit is 0 . The exponent is 01101110 . The fractional part is 01 followed by 21 zeros. The IEEE form, showing the fields, is

0	01111110	01000000000000000000000

sign exponent fractional part

or $3F200000_{16}$.

IEEE storage of the number 0 is interesting. By definition, 0 is stored in by putting 31 zeros in the exponent and fractional fields. The sign bit is not specified. This allows for positive 0 and negative 0 (1 followed by 31 zeros)!